Please amend the present application as follows:

Claims

The following is a copy of Applicant's claims that identifies language being added with underlining ("___") and language being deleted with strikethrough ("—"), as is applicable:

1-22. Cancelled.

23. (Previously presented) A method for passing information to a post-compile-time software application, comprising the steps of:

compiling a plurality of blocks of code, including finding one or more unused bits in an instruction in one of the plurality of blocks of code, and using the one or more unused bits to encode information without defining new instructions or opcodes; and

communicating the information to the post-compile-time software application for use by the post-compile-time software application.

24. (Previously presented) The method of claim 23, wherein the information identifies whether certain registers are live.

25. (Previously presented) The method of claim 23, wherein the post-compile-time software application comprises a dynamic optimizer.

26. (Previously presented) The method of claim 23, wherein the instruction is a no-operation (NOP) instruction.

27. (Previously presented) The method of claim 23, wherein the information is used by the post-compile-time software application to determine whether certain registers are live.

28. (Previously presented) The method of claim 23, wherein the information is encoded as a bit vector.

29. (Previously presented) The method of claim 23, wherein the step of using the one or more unused bits to encode information, comprises:

determining which of a plurality of registers are live in said one of the plurality of blocks of code;

creating within the instruction a register-usage bit-vector having a plurality of register-usage bits; and

setting one of the plurality of register-usage bits for each one of the plurality of registers that are live.

30. (Previously presented) A system comprising:

a compiler that is configured to compile a plurality of blocks of code, the compiler including a code annotator that is configured to find one or more unused bits in an instruction in one of the plurality of blocks of code that are being compiled by the compiler, and to encode information in the one or more unused bits without defining new instructions or opcodes, the information being configured to be used by a post-compile-time software application; and

a processor configured to execute the compiler.

31. (Previously presented) The system of claim 30, wherein the information identifies whether certain registers in said one of the plurality of blocks of code are live.

32. (Previously presented) The system of claim 30, wherein the post-compile-time software application comprises a dynamic optimizer.

33. (Previously presented) The system of claim 30, wherein the instruction is a no-operation (NOP) instruction.

34. (Previously presented) The system of claim 30, wherein the post-compile-time software application is configured to use the information to determine whether certain registers are live.

35. (Previously presented) The system of claim 30, wherein the information is encoded as a bit vector.

36. (Previously presented) A system comprising:

means for compiling a plurality of blocks of code, including finding one or more unused bits in an instruction in one of the plurality of blocks of code, and using the one or more unused bits to pass information to a post-compile-time software application without defining new instructions or opcodes; and

means for executing the means for compiling and the post-compile time software application.

37. (Previously presented) The system of claim 36, wherein the information identifies whether certain registers are live.

38. (Previously presented) The system of claim 36, wherein the post-compile-time software application comprises a dynamic optimizer.

39. (Previously presented) The system of claim 36, wherein the instruction is a no-operation (NOP) instruction.

40. (Previously presented) A method for compiling, comprising the steps of:

finding one or more unused bits in an instruction in one of a plurality of blocks of code that are being compiled; and

encoding information in the one or more unused bits without defining new instructions or opcodes, wherein the information is used by a post-compile-time software application.

41. (Previously presented) The method of claim 40, wherein the information identifies whether certain registers are live.

42. (Previously presented) The method of claim 40, wherein the post-compile-time software application comprises a dynamic optimizer.

43. (Previously presented) The method of claim 40, wherein the instruction is a no-operation (NOP) instruction.

44. (Previously presented) The method of claim 40, further comprising: using the information by the post-compile-time software application to determine whether certain registers are live.

45. (Previously presented) The method of claim 40, wherein the information is encoded as a bit vector.

46. (Previously presented) A method for passing information to a post-compile-time software application, comprising the steps of:

compiling a plurality of blocks of code; and

during the step of compiling, finding one or more unused bits in an instruction in one of the plurality of blocks of code, wherein the one or more unused bits are used to pass information to the post-compile-time software application without defining new instructions or opcodes.

47. (Previously presented) The method of claim 23, wherein the information identifies whether certain registers are live.

48. (Previously presented) The method of claim 23, wherein the instruction is a no-operation (NOP) instruction.

49. (Previously presented) A computer system comprising:

memory configured to store software;

a processor that is coupled to the memory and that is configured to execute software stored in the memory;

a compiler that is stored in the memory and that is configured to compile blocks of code, to find unused bits in an instruction in one of the blocks of code, and to encode information as a bit vector in the unused bits; and

a post-compile-time software application that is stored in the memory and that is configured to use the information to modify the compiled blocks of code.

50. (Previously presented) The system of claim 49, wherein the instruction is a no-operation (NOP) instruction.

51. (Previously presented) The system of claim 49, wherein the post-compile-time software application is configured to use the information to determine whether certain registers are live.

52. (Previously presented) The system of claim 49, wherein the information identifies whether certain registers in said one of the blocks of code are live.

53. (Previously presented) The system of claim 49, wherein the post-compile-time software application comprises a dynamic optimizer.

54. (Previously presented) A method that is implemented by software, comprising the steps of:

storing a post-compile-time software application in memory that is coupled to a processor;

compiling blocks of code, including finding unused bits in an instruction in one of the blocks of code, and encoding information as a bit vector in the unused bits; and

modifying the compiled blocks of code by the post-compile-time software application, wherein a configuration of the modified and compiled blocks of code is responsive to the information.

55. (Previously presented) The method of claim 54, wherein the instruction is a no-operation (NOP) instruction.

56. (Previously presented) The method of claim 54, wherein the post-compile-time software application is configured to use the information to determine whether certain registers are live.

57. (Previously presented) The method of claim 54, wherein the information identifies whether certain registers in said one of the blocks of code are live.

58. (Previously presented) The method of claim 54, wherein the post-compile-time software application comprises a dynamic optimizer.

59. (Previously presented) A computer system comprising:

a compiler configured to compile blocks of code, to find unused bits in a NOP instruction in one of the blocks of code, and to encode information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live; and

a dynamic optimizer that is configured to optimize the compiled blocks of code, wherein a configuration of the optimized and compiled blocks of code is responsive to the information.

60. (Previously presented) The computer system of claim 59, further comprising:

a register usage comparator configured to identify live registers responsive to the information.

61. (Previously presented) A method comprising:

compiling blocks of code, including finding unused bits in a NOP instruction in one of the blocks of code, and encoding information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live; and

optimizing the compiled blocks of code, wherein a configuration of the optimized and compiled blocks of code is responsive to the information.

62. (Previously presented) The method of claim 61, further comprising: identifying live registers responsive to the information.

63. (Previously presented) The method of claim 61, wherein the information is encoded without defining new instructions or opcodes.